INSTITUTO FEDERAL Bahia Campus Vitória da Conquista	**LISTA DE EXERCÍCIOS 05**	
	CURSO: Bacharelado em Sistemas de Informação	**MODALIDADE:** Ensino Superior
	MÓDULO/SEMESTRE/SÉRIE: 3º	**PERÍODO LETIVO:** 2024.1
	DISCIPLINA: Linguagem de Programação II	**CLASSE:** 20241.3.119.1N
	DOCENTE: Alexandro dos Santos Silva	

INSTRUÇÕES

- Para resolução das questões abaixo, será admitido o uso apenas da sintaxe adotada para escrita de programas em Java.

1. Considere a classe abaixo, para fins de encapsulamento de dados típicos de livros.

```
01  package lingprog2.lista05.questao01;
02
03  //Encapsulamento de dados típicos de livro
04  public class Livro {
05
06     // atributos
07     private String titulo;       // título
08     private String autores;      // lista de autores separados por vírgula
09     private int edicao;          // número de edição (inteiro igual ou superior a 1)
10     private int ano;             // ano de edição (inteiro positivo de 4 dígitos)
11
12     // método construtor
13     public Livro(String titulo, String autores, int edicao, int ano) {
14        this.titulo = titulo;
15        this.autores = autores;
16        this.edicao = edicao;
17        this.ano = ano;
18     }
19
20     // métodos getters e setters
21     public String getTitulo() {
22        return titulo;
23     }
24
25     public void setTitulo(String titulo) {
26        this.titulo = titulo;
27     }
28
29     public String getAutores() {
30        return autores;
31     }
32
33     public void setAutores(String autores) {
34        this.autores = autores;
35     }
36
37     public int getEdicao() {
38        return edicao;
39     }
40
41     public void setEdicao(int edicao) {
42        this.edicao = edicao;
43     }
44
45     public int getAno() {
46        return ano;
47     }
48
49     public void setAno(int ano) {
50        this.ano = ano;
       }
50
51     // retorno de descrição de livro considerando-se seu estado atual
52     @Override
53     public String toString() {
54        return "Livro [titulo=" + titulo + ", autores=" + autores + ", edicao=" + edicao +
                 ", ano=" + ano + "]";
55     }
56
57  }
```

Implemente uma classe utilitária que disponha de método estático **`main`**, no qual seja manipulada uma pilha de instâncias da classe **`Livro`** através de alguma classe da biblioteca de coleções da linguagem Java que implemente a interface **`java.util.Deque<E>`**. Com base no principio basilar de qualquer pilha (último item a ser inserido é o primeiro item a ser removido), após inserção de 5 (cinco) livros naquela pilha, o antepenúltimo item inserido deve ser removido mantendo-se, após

isso, os dois últimos itens inseridos no topo da pilha (para tal, recomenda-se o uso de uma coleção auxiliar de armazenamento temporário de objetos que serão desempilhados e, posteriormente, empilhados novamente).

Observação: para a definição dos valores dos campos de instância dos objetos da classe **`Livro`**, use operações de entrada de dados. Além disso, certifique-se da exibição de todos os itens da pilha antes e após remoção do antepenúltimo objeto instanciado, pela invocação do método **`toString`** (herdado, pelos objetos que implementam a interface **`java.util.Deque<E>`**, da classe **`java.lang.Object`**).

2. A biblioteca de coleções da linguagem Java fornece uma interface específica para operações de inserção, extração e consulta típicas de uma *fila* (**`java.util.Queue<E>`**) e de algumas implementações desta interface, a exemplo de **`java.util.ArrayDeque<E>`**. Para aplicação prática desta implementação, considere o conceito de *fator primo*, conhecido como, para um dado número inteiro $n > 1$, o menor inteiro $d > 1$ que divide n. É possível determinar a *fatoração prima* de n achando-se o fator primo d e substituindo n pelo quociente n / d, repetindo essa operação até que n seja igual a 1. Implemente uma classe utilitária que disponha de método estático **`main`**, no qual seja fornecido um número inteiro após o que seus fatores primos são identificados e enfileirados; após o término do enfileiramento, eles deverão ser removidos para exibição da fatoração prima, na forma $d_1 \times d_2 \times ... \times d_n$. Por exemplo, para o número 3.960, a fatoração prima corresponderia a $2 \times 2 \times 2 \times 3 \times 3 \times 5 \times 11$.

3. Considere a classe abaixo, para fins de encapsulamento de dados típicos de contatos telefônicos.

```
01  package lingprog2.lista05.questao03;
02
03  // Encapsulamento de dados típicos de contato telefônico
04  public class ContatoTelefonico {
05
06     // constantes para fins de indicação de categorias de contatos
07     public static final int FAMILIAR = 0;
08     public static final int PROFISSIONAL = 1;
09     public static final int OUTROS = 2;
10
11     // atributos
12     private String nome;          // primeiro nome
13     private String sobrenome;     // sobrenome
14     private String email;         // endereço de e-mail
15     private int codigoPais;       // código telefônico de país
16     private String telefone;      // número telefônico
17     private int categoria;        // categoria (familiar, profissional ou outros)
18
19     // método construtor
20     public ContatoTelefonico(String nome, String sobrenome, String email, int codigoPais,
                                String telefone, int categoria) throws IllegalArgumentException {
21        // se categoria não corresponder a alguma das constantes definidas anteriormente...
22        if (categoria != FAMILIAR && categoria != PROFISSIONAL && categoria != OUTROS) {
23           // lançamento de exceção
24           throw new IllegalArgumentException("Tipo de contato inválido!");
25        }
26
27        this.nome = nome;
28        this.sobrenome = sobrenome;
29        this.email = email;
30        this.codigoPais = codigoPais;
31        this.telefone = telefone;
32        this.categoria = categoria;
33     }
34
35     // métodos getters e setters
36     public String getNome() {
37        return nome;
38     }
39
40     public void setNome(String nome) {
41        this.nome = nome;
42     }
43
44     public String getSobrenome() {
45        return sobrenome;
46     }
47
48     public void setSobrenome(String sobrenome) {
49        this.sobrenome = sobrenome;
50     }
50
51     public String getEmail() {
52        return email;
53     }
54
55     public void setEmail(String email) {
```

```
56          this.email = email;
57      }
58
59      public int getCodigoPais() {
60          return codigoPais;
61      }
62
63      public void setCodigoPais(int codigoPais) {
64          this.codigoPais = codigoPais;
65      }
66
67      public String getTelefone() {
68          return telefone;
69      }
70
71      public void setTelefone(String telefone) {
72          this.telefone = telefone;
73      }
74
75      public int getCategoria() {
76          return categoria;
77      }
78
79      public void setCategoria(int categoria) {
80          this.categoria = categoria;
81      }
82
83      // retorno de descrição de contato telefônico considerando-se seu estado atual
84      @Override
85      public String toString() {
86          return "ContatoTelefonico [nome=" + nome + ", sobrenome=" + sobrenome +
                    ", email=" + email + ", codigoPais=" + codigoPais + ", telefone=" + telefone +
                    ", categoria=" + categoria + "]";
87      }
88
89  }
```

Implemente classe utilitária de nome **`ContatoTelefonicoUtil`** de modo que a mesma disponha de método estático **`main`** para fins de manipulação de lista de instâncias da classe **`ContatoTelefonico`** utilizando-se de alguma classe da biblioteca de coleções da linguagem Java que implemente a interface **`java.util.List<E>`**. Deverá ser permitido a qualquer momento executar uma da seguintes operações: a) inserção de novo contato telefônico; b) listagem de contatos telefônicos de determinado país considerando-se código telefônico daquele país a ser fornecido pelo usuário; c) listagem de percentuais de contatos telefônicos por categoria; e d) encerramento do programa.

Observação: quando da inserção de novo contato telefônico, certifique-se da impossibilidade de estarem inseridos na lista 2 (dois) ou mais contatos com mesmo nome, sobrenome, código de país e número telefônico.

4. Readeque a resolução da questão anterior em relação aos seguintes aspectos:
   a) Sobrescrita, em classe **`ContatoTelefonico`**, de método **`equals`** (herdado de forma implícita da classe **`java.lang.Object`**) considerando-se que dois ou mais contatos telefônicos sejam idênticos se possuírem mesmo nome, sobrenome, código de país e número telefônico;
   b) Substituição, em classe **`ContatoTelefonicoUtil`**, de implementação da interface **`java.util.List<E>`** por alguma implementação da interface **`java.util.Set<E>`**.

5. Tabelas de dispersão (também conhecidas como tabelas *hash*) armazenam elementos com base no valor absoluto de suas chaves e em técnicas de tratamento de colisões. As funções de dispersão transformam chaves ou valores em endereços base da tabela, ao passo que o tratamento de colisões resolve conflitos em casos em que mais de uma chave ou valor é mapeada para um mesmo endereço da tabela. Suponha que uma aplicação utilize uma tabela de dispersão com 23 endereços (índices de 0 a 22) e empregue a função de dispersão

   `h(x) = x mod 23`, em que **`x`** representa a chave ou valor do elemento cujo endereço deseja-se computar

   Implemente uma classe utilitária que disponha de método estático **`main`**, no qual seja manipulada um mapa de *hash* usando-se a classe **`java.util.HashMap<K, V>`**. Deverá ser permitido a qualquer momento executar uma da seguintes operações: a) inserção de novo valor numérico inteiro no mapa de *hash*; b) listagem de valores numéricos inteiros associados a determinado endereço ou índice (entre 0 e 22) e já inseridos no mapa de *hash*; e c) encerramento do programa. Assuma que o mapa de *hash* trate colisões por meio de encadeamento exterior, associando a cada endereço ou índice de dispersão um objeto da classe **`java.util.ArrayList<java.lang.Integer>`**.

6. Considere a inserção de 30.000 números inteiros em objeto da classe **`java.util.ArrayList`** e, após isso, pesquisa desses mesmos números naquela coleção, conforme implementação que se segue abaixo (o tempo de processamento das operações é cronometrado para fins de aferição):

```
01  package lingprog2.lista05.questao06;
02
03  import java.util.ArrayList;
04  import java.util.Collection;
05
06  public class TestePerformance {
07
08          public static void main(String[] args) {
09                  System.out.println("Iniciando teste...");
10
11                  long inicio = System.currentTimeMillis();
12
13                  Collection<Integer> colecao = new ArrayList<Integer>();
14
15                  int total = 30000;
16
17                  // Inserção de 30.000 números
18                  for (int i = 0; i < total; i++) {
19                          colecao.add(i);
20                  }
21
22                  // Pesquisa de 30.000 números
23                  for (int i = 0; i < total; i++) {
24                          colecao.contains(i);
25                  }
26
27                  long termino = System.currentTimeMillis();
28
29                  long tempo = termino - inicio;
30
31                  System.out.println("Tempo de processamento: " + tempo + " ms");
32          }
33
34  }
```

Após substituir o objeto da classe **`java.util.ArrayList`** por um objeto da classe **`java.util.HashSet`**, o tempo de processamento aumentará ou diminuirá? O que é lento? A inserção de 30 mil elementos ou as 30 mil buscas? Para tal, compute o tempo gasto em cada bloco de repetição **`for`** separadamente.